Δομές Δεδομένων

ΑΤΔ - Χάρτες

Α. Συμβώνης
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών
Τομέας Μαθηματικών

Χάρτης
[Map]

Χάρτης [map]

- Ένας χάρτης μοντελοποιεί μια συλλογή από εγγραφές τύπου <κλειδί [key], τιμή [value]> η οποία είναι αναζητήσιμη με βάση το κλειδί.
- Οι βασικές λειτουργίες ενός χάρτη είναι:
  - αναζήτηση εγγραφής
  - εισαγωγή εγγραφής
  - διαγραφή εγγραφής
- Πολλαπλές εγγραφές με το ίδιο κλειδί δεν επιτρέπονται
- Εφαρμογές
  - Τηλεφωνικός κατάλογος
  - Βάση δεδομένων βαθμολογίας φοιτητών

Χάρτης [map]

- Ο ΑΤΔ Χάρτης
  - `put(k,v)`: Εισάγει μια εγγραφή (`k`, `v`) στον χάρτη. Εάν το κλειδί `k` δεν υπάρχει ήδη στον χαρτη, τότε επιστρέφει `null`, διαφορετικά επιστρέφει την προηγούμενη τιμή που αντιστοιχούσε στο κλειδί `k`
  - `get(k)`: Εάν ο χάρτης περιέχει μια εγγραφή με κλειδί `k`, επιστρέφει την αντίστοιχη τιμή, διαφορετικά επιστρέφει `null`
  - `remove(k)`: Εάν ο χάρτης περιέχει μια εγγραφή με κλειδί `k`, την αφαιρεί και επιστρέφει την αντίστοιχη τιμή, διαφορετικά επιστρέφει `null`
  - `size()`: Το πλήθος των καταχωρήσεων που περιέχει ο χάρτης
  - `isEmpty()`: Υποδηλώνει εάν ο χάρτης είναι άδειος ή όχι
  - `entrySet()`: Επιστρέφει μια `iterable` συλλογή από τις εγγραφές του χάρτη
  - `keySet()`: Επιστρέφει μια `iterable` συλλογή από τα κλειδιά του χάρτη
  - `values()`: Επιστρέφει έναν `iterator` των τιμών του χάρτη

Χάρτης [map]

- Η διαπροσωπία `Map`

```
1  public interface Map<K,V> {
2    V put(K key, V value);
3    V get(K key);
4    V remove(K key);
5    int size();
6    boolean isEmpty();
7    Iterable<Entry<K,V>> entrySet();
8    Iterable<K> keySet();
9    Iterable<V> values();
10 }
```

```
2  public interface Entry<K,V> {
3    K getKey();
4    V getValue();
5  }
```

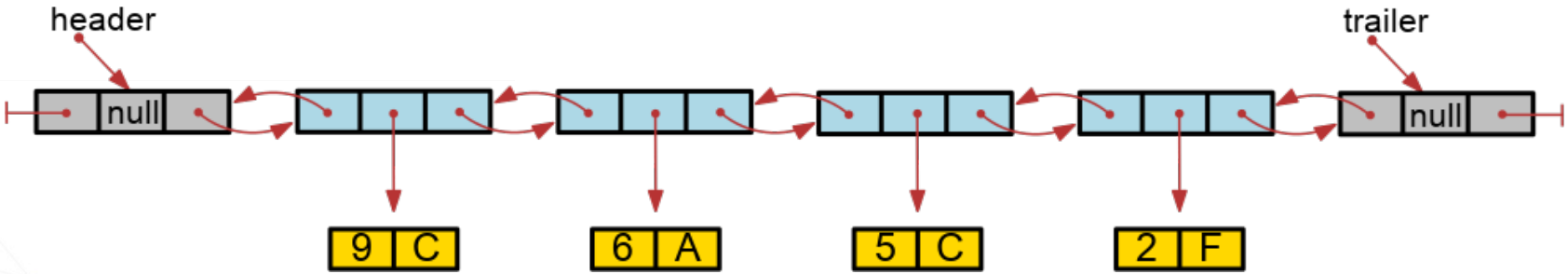
Χάρτης [map]

Παράδειγμα: Χρήση `Map`

Method	Return value	Περιεχόμενα χάρτη
`isEmpty()`	`true`	`()`
`put(5,A)`	`null`	`( [5,A] )`
`put(7,B)`	`null`	`( [5,A], [7,B] )`
`put(2,C)`	`null`	`( [5,A], [7,B], [2,C] )`
`isEmpty()`	`false`	`( [5,A], [7,B], [2,C] )`
`put(2,E)`	`C`	`( [5,A], [7,B], [2,E] )`
`get(7)`	`B`	`( [5,A], [7,B], [2,E] )`
`get(4)`	`null`	`( [5,A], [7,B], [2,E] )`
`size()`	`3`	`( [5,A], [7,B], [2,E] )`
`remove(4)`	`null`	`( [5,A], [7,B], [2,E] )`
`remove(5)`	`A`	`( [7,B], [2,E] )`
`get(5)`	`null`	`( [7,B], [2,E] )`

Χάρτης [map]

- Υλοποίηση του ΑΤΔ Χάρτη με λίστα
  - Χρησιμοποιούμε μια μη-ταξινομημένη λίστα `S`
  - Αποθηκεύουμε τα στοιχεία του χάρτη στη λίστα `S` (διπλά συνδεδεμένη λίστα)



Χάρτης [map]

- Ο αλγόριθμος `get(k)`

```
Algorithm get(k):
    iter = s.positions()    //iter is an iterator of Positions in s
    while iter.hasNext() do
        p = iter.next()     //the next Position in iter
        if p.element().getKey() == k then
            return p.element().getValue()
    return null             //there is no entry with key equal to k
```

Χάρτης [map]

- Ο αλγόριθμος `put(k,v)`

Algorithm `put(k,v)`:

```
iter = s.positions()
while iter.hasNext() do
    p = iter.next()
    if p.element().getKey() == k then
        t = p.element().getVaue()
        s.set(p,(k,v))
        return t              //return the old value
s.addLast((k,v))
return null
```

Χάρτης [map]

- Ο αλγόριθμος `remove(k)`

```
Algorithm remove(k):
    iter = s.positions()
    while iter.hasNext() do
        p = iter.next()
        if p.element().getKey() == k  then
            t = p.element().getVaue()
            s.remove(p)
            return t                    //return the removed value
    return null                         //there is no entry with key equal to k
```

Χάρτης [map]

Πολυπλοκότητα:

	Υλοποίηση
Μέθοδος	Λίστα
`put(k,v)`	`O(n)`
`get(k)`	`O(n)`
`remove(k)`	`O(n)`
`size()`	`O(1)`
`isEmpty()`	`O(1)`

- Η υλοποίηση με μη-ταξινομημένη λίστα είναι αποδοτική μόνο για μικρού μεγέθους χάρτες

Πίνακας Κατακερματισμού
[Hash Table]

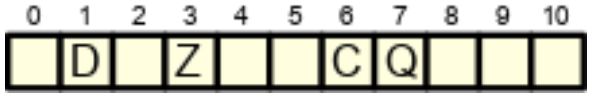
Πίνακας Κατακερματισμού [Hash Table]

- Υπενθύμιση ΑΤΔ Χάρτης
  - `put(k,v)`: Εισάγει μια εγγραφή (`k`, `v`) στον χάρτη. Εάν το κλειδί `k` δεν υπάρχει ήδη στον χαρτη, τότε επιστρέφει `null`, διαφορετικά επιστρέφει την προηγούμενη τιμή που αντιστοιχούσε στο κλειδί `k`
  - `get(k)`: Εάν ο χάρτης περιέχει μια εγγραφή με κλειδί `k`, επιστρέφει την αντίστοιχη τιμή, διαφορετικά επιστρέφει `null`
  - `remove(k)`: Εάν ο χάρτης περιέχει μια εγγραφή με κλειδί `k`, την αφαιρεί και επιστρέφει την αντίστοιχη τιμή, διαφορετικά επιστρέφει `null`
  - `size()`: Το πλήθος των καταχωρήσεων που περιέχει ο χάρτης
  - `isEmpty()`: Υποδηλώνει εάν ο χάρτης είναι άδειος ή όχι
  - `entrySet()`: Επιστρέφει μια `iterable` συλλογή από τις εγγραφές του χάρτη
  - `keySet()`: Επιστρέφει μια `iterable` συλλογή από τα κλειδιά του χάρτη
  - `values()`: Επιστρέφει έναν `iterator` των τιμών του χάρτη

Πίνακας Κατακερματισμού [Hash Table]

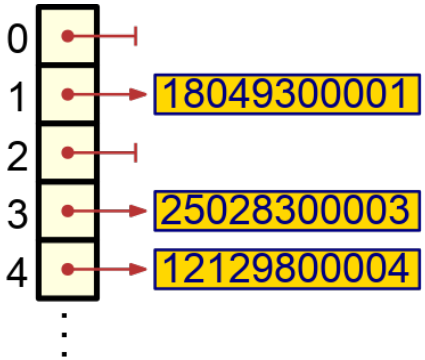
- Ένας χάρτης `m` υποστηρίζει, διαισθητικά, τη χρήση των κλειδιών ως δείκτες με μία σύνταξη της μορφής `m[k]`

Παράδειγμα: Θεωρήστε έναν χάρτη με `n` στοιχεία ο οποίος χρησιμοποιεί κλειδιά τα οποία είναι ακεραίοι, με εύρος τιμών `[0,N-1]` για κάποιο $N \geq n$



- Τι γίνεται όμως αν τα κλειδιά δεν είναι ακέραιοι;
  - Μπορούμε να χρησιμοποιήσουμε συναρτήσεις κατακερματισμού [hash functions] για να αντιστοιχίσουμε τα κλειδιά σε ακεραίους και, κατόπιν, να τους χρησιμοποιήσουμε ως δείκτες σε ένα διάνυσμα.

Παράδειγμα: Χρήση των τελευταίων 5 ψηφία του ΑΜΚΑ



Πίνακας Κατακερματισμού [Hash Table]

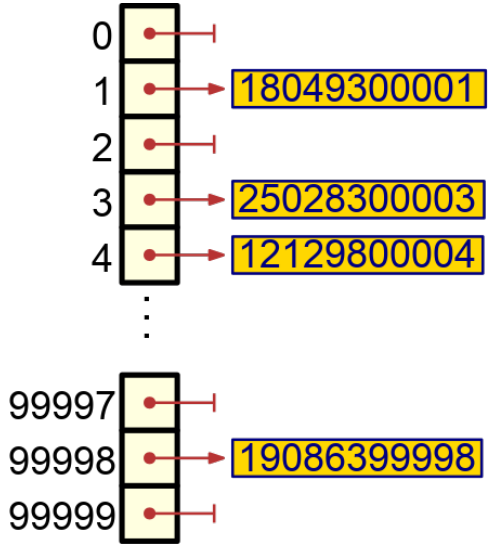
Ορισμοί

- Μία συνάρτηση κατακερματισμού [hash function] `h` αντιστοιχεί κλειδιά ενός συγκεκριμένου τύπου σε ακεραίους ενός σταθερού εύρους `[0,N-1]`
  - Παράδειγμα: Συνάρτηση κατακερματισμού ακεραίων κλειδιών `h(x) = x mod N`
- Η ακέραια τιμή `h(x)` ονομάζεται τιμή κατακερματισμού του κλειδιού `x`
- Ένας πίνακας κατακερματισμού για ένα συγκεκριμένο τύπο κλειδιού αποτελείται από:
  - Μια συνάρτηση κατακερματισμού
  - Ένας διάνυσμα (ή αλλιώς, πίνακα) μεγέθους `N`
- Όταν υλοποιούμε ένα χάρτη με πίνακα κατακερματισμού, στόχος μας είναι να αποθηκεύσουμε το στοιχείο `(k,v)` στη θέση `i = h(k)`

Πίνακας Κατακερματισμού [Hash Table]

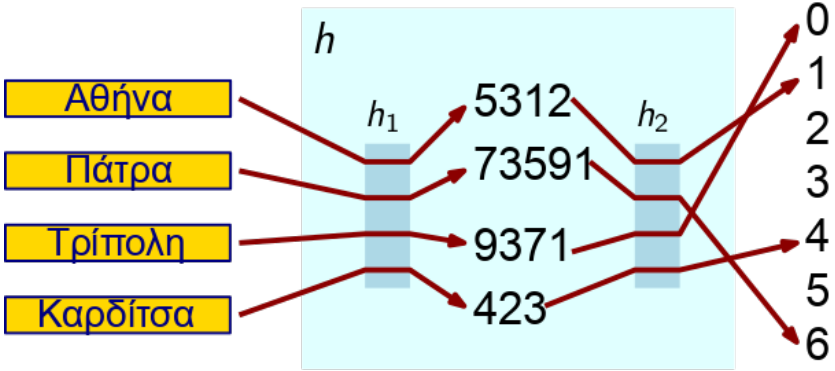
Παράδειγμα

- Σχεδιάζουμε ένα πίνακα κατακερματισμού για ένα χάρτη, ο οποίος αποθηκεύει εγγραφές όπως (ΑΜΚΑ, ονοματεπώνυμο), όπου ο ΑΜΚΑ είναι ένας θετικός ακέραιος 11 ψηφιών
- Ο πίνακας κατακερματισμού χρησιμοποιεί ένα διάνυσμα με `N = 100.000 θέσεις` και συνάρτηση κατακερματισμού: `h(x)= x % 100.000`, δηλαδή, τα τελευταία 5 ψηφία του `x`.



Πίνακας Κατακερματισμού [Hash Table]

- Μία συνάρτηση κατακερματισμού συνήθως προσδιορίζεται ως ο συνδυασμός δύο συναρτήσεων:
  - Κώδικα κατακερματισμού: $h_1: keys \rightarrow integers$
  - Συνάρτηση συμπίεσης: $h_2: integers \rightarrow [0, N-1]$
- Ο κώδικας κατακερματισμού εφαρμόζεται πρώτος και η συνάρτηση συμπίεσης εφαρμόζεται δεύτερη στο αποτέλεσμα, δηλαδή: $h(x) = h_2\big(h_1(x)\big)$
- Ο στόχος της συνάρτησης κατανομής είναι να "διασκορπίσει" τα κλειδιά με ένα φαινομενικά τυχαίο τρόπο



Πίνακας Κατακερματισμού [Hash Table]

- Κώδικες κατακεμαρτισμού
  - Η διεύθυνση μνήμης [memory address]
    - Επανερμηνεύουμε τη διεύθυνση μνήμης του αντικειμένου του κλειδιού ως ακέραιο (προκαθορισμένη μέθοδος `hashCode()` όλων των Java αντικειμένων)
    - Γενικά είναι καλός, εκτός από τις περιπτώσεις όπου τα κλειδιά είναι αριθμοί ή String
    - Προσοχή: Όμοια αλλά όχι ταυτόσημα αντικείμενα συνήθως έχουν διαφορετικό `hashCode()`
  - Μετατροπή τύπου σε ακέραιο [integer cast]
    - Επανερμηνεύουμε τα bits του κλειδιού σαν ακέραιο
    - Κατάλληλο για κλειδιά μήκους μικρότερου ή ίσου του πλήθους των bits των ακέραιων τύπων της Java (πχ byte, short, int, float)
  - Πρόσθεση συνιστωσών [component sum]
    - Χωρίζουμε τα bits του κλειδιού σε τμήματα σταθερού μήκους (πχ 16 ή 32 bits) και αθροίζουμε τα τμήματα (αγνοώντας τις υπερχειλίσεις)
    - Κατάλληλο για αριθμητικά κλειδιά σταθερού μήκους μεγαλύτερου ή ίσου του πλήθους των bits των ακέραιων τύπων της Java (πχ long, double)

Πίνακας Κατακερματισμού [Hash Table]

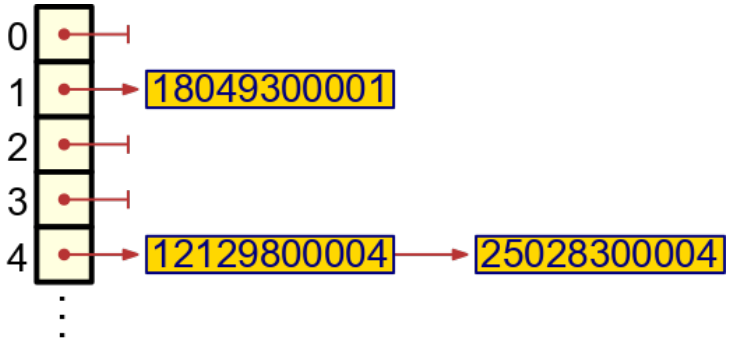
- Κώδικες κατακεμαρτισμού (*συνέχεια*)
  - Πολυωνυμική συσσώρευση [polynomial accumulation]
    - Χωρίζουμε τα bits του κλειδιού σε μια ακολουθία από τμήματα σταθερού μήκους (πχ 8, 16 ή 32 bits) $a_0 a_1 \dots a_{n-1}$
    - Υπολογίζουμε το πολυώνυμο $p(z) = a_0 + a_1 z + a_2 z^2 + \cdots + a_{n-1} z^{n-1}$ σε μία σταθερή τιμή z, αγνοώντας τις υπερχειλίσεις
    - Ειδικά κατάλληλο για τα String (πχ επιλέγοντας $z = 33$ προκύπτουν το πολύ 6 «συγκρούσεις» σε ένα σύνολο 50.000 λέξεων)
    - Το πολυώνυμο $p(z)$ μπορεί να υπολογιστεί σε χρόνο $O(n)$ χρησιμοποιώντας το κανόνα του Horner σε χρόνο $O(n)$:
      - $p_0(z) = a_{n-1}$
      - $p_i(z) = a_{n-i-1} + z \cdot p_{i-1}(z), \quad i = 1,2,\dots,n-1$
      - Τελικά έχουμε: $p(z) = p_{n-1}(z)$

Πίνακας Κατακερματισμού [Hash Table]

- Συνάρτησεις συμπίεσης
  - Διαίρεση
    - $h_2(x) = x \bmod N$
    - Το μέγεθος N του πίνακα κατακερματισμού επιλέγεται συνήθως να είναι πρώτος αριθμός
  - Πολλαπλασιασμός, Πρόσθεση και Διαίρεση (MAD)
    - $h_2(x) = (ax + b) \bmod N$
    - Οι αριθμοί a και b είναι μη αρνητικοί ακέραιοι τέτοιοι ώστε: $a \bmod N \neq 0$, διαφορετικά κάθε ακέραιος y θα αντιστοιχούταν στην ίδια τιμή b

Πίνακας Κατακερματισμού: χρήση αλυσίδων

- Διαχείριση συγκρούσεων
  - Μια σύγκρουση συμβαίνει όταν δύο κλειδιά αντιστοιχίζονται στην ίδια τιμή κατακερματισμού
  - Χρήση αλυσίδων [chaining]:
    - Το κάθε κελί του πίνακα κατακερματισμού δείχνει σε μια συνδεδεμένη λίστα (αλυσίδα) από εγγραφές που αντιστιχούν σε αυτή τη τιμή
    - Απλή λύση, αλλά απαιτεί επιπλέον μνήμη έξω από τον πίνακα κατακερματισμού



Πίνακας Κατακερματισμού: χρήση αλυσίδων

- Τυπικές υλοποιήσεις για χάρτη τύπου πίνακα κατερματισμού με λίστες σε κάθε κελί

```
Algorithm put(k,v):
    t = data[h(k)].put(k,v)
    if t == null then   //k is a new key
        n = n + 1
    return t

Algorithm get(k):
    return data[h(k)].get(k)

Algorithm remove(k):
    t = data[h(k)].remove(k)
    if t != null then   //k was found
        n = n - 1
    return t
```

Πίνακας Κατακερματισμού: χρήση αλυσίδων

Ανάλυση

- Ομοιόμορφη Υπόθεση
  - Η συνάρτηση κατακερματισμού κατανέμει ομοιόμορφα το σύμπαν στις διαθέσιμες θέσεις.
  - Όλα τα μέλη του σύμπαντος έχουν την ίδια πιθανότητα να αποτελέσουν έντελο κάποιας πράξης

- Παράγοντας φορτίου [load factor] $\alpha = n/N$

Λήμμα: Σε μία δομή κατακερματισμού με αλυσίδες n στοιχείων:
- Το μέσο μήκος των αλυσίδων είναι α με πιθανότητα να τείνει στο 1.
- Μία αναζήτηση (επιτυχημένη ή αποτυχημένη) κοστίζει $O(1+\alpha)$ χρόνο.

Θεώρημα: Μία ακολουθία από n ενθέσεις, διαγραφές, και αναζητήσεις σε ένα πίνακα κατακερματισμού με χρήση αλυσίδων απαιτεί χρόνο $O(n(1+\alpha/2)$.

Θεώρημα: Σε ένα πίνακα κατακερματισμού με χρήση αλυσίδων που έχει n εγγραφές και παράγοντα φορτίου $\alpha < 1$, το μέσο μήκος της μεγαλύτερης αλυσίδας είναι $O\left(\frac{\log n}{\log \log n}\right)$

Πίνακας Κατακερματισμού: Γραμμική εξέταση

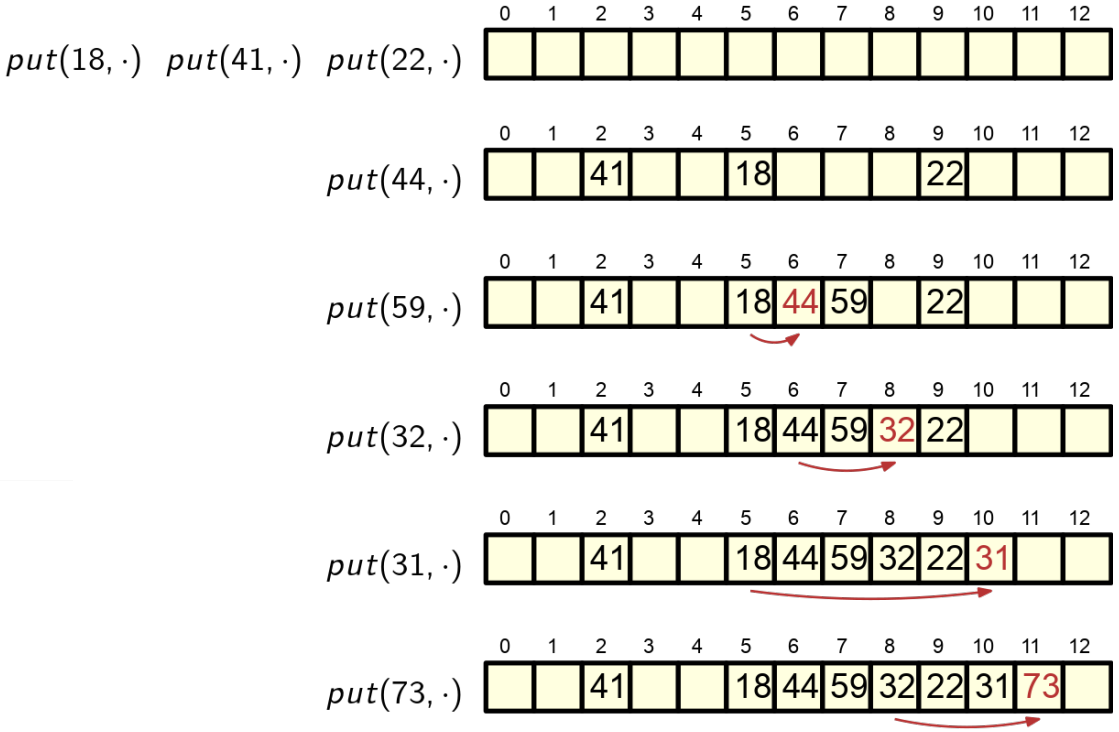
Γραμμική Εξέταση [linear probing]

- Προσέγγιση «Ανοικτών διευθύνσεων» [open addressing]
  - Οι συγκρουόμενες εγγραφές τοποθετούνται σε διαφορετικά κελιά του πίνακα κατακερματισμού
- Γραμμική εξέταση [linear probing]
  - Διαχειρίζεται τις συγκρούσεις τοποθετώντας τη συγκρουόμενη εγγραφή στο επόμενο (κυκλικά) διαθέσιμο κελί του πίνακα κατακερματισμού
- Κάθε κελί του πίνακα που εξετάζεται αναφέρεται ως "προσπάθεια/διερεύνηση/εξέταση" [probe]
- Οι συγκρουόμενες εγγραφές συσσωρεύονται με συνέπεια οι μελλοντικές συγκρούσεις να δημιουργούν μεγαλύτερες ακολουθίες από "προσπάθειες"
- Κενωθείσες/ανενεργές θέσεις [defunct positions]: Θέσεις του πίνακα οι οποίες ήταν κατειλημμένες και εκκενώθηκαν μετά από την διαγραφή της εγγραφής που φιλοξενούσαν.

Πίνακας Κατακερματισμού: Γραμμική εξέταση

Παράδειγμα

- $h(x) = x \bmod 13$

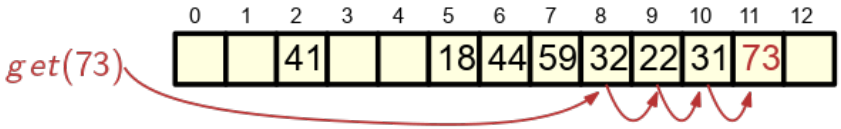


Πίνακας Κατακερματισμού: Γραμμική εξέταση

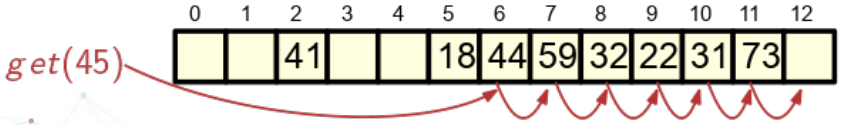
- Αναζήτηση στοιχείου
  - Υποθέτουμε πίνακα κατακερματισμού ο οποίος χρησιμοποιεί γραμμική εξέταση [linear probing]
    - Ξεκινάμε την αναζήτηση από το κελί $h(k)$
    - Εξετάζουμε διαδοχικά τα κελιά μέχρι να συμβεί ένα από τα παρακάτω:
      - Βρέθηκε μία εγγραφή με κλειδί k
      - Βρέθηκε ένα άδειο κελί
      - N κελιά διερευνήθηκαν ανεπιτυχώς

Παράδειγμα

- $h(x) = x \bmod 13$



- Επιστρέφει την τιμή της εγγραφής στη θέση 11



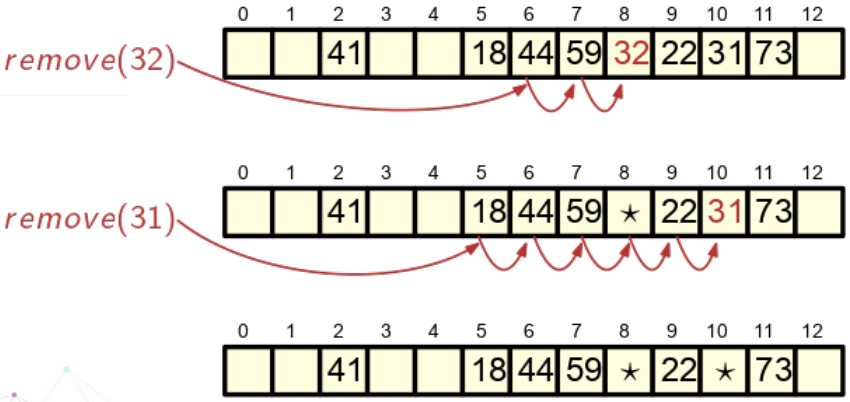
- Επιστρέφει null

Πίνακας Κατακερματισμού: Γραμμική εξέταση

- Αφαίρεση στοιχείου
  - Εισάγουμε μία ειδική εγγραφή, την 'ανενεργή' [**defunct**], η οποία αντικαθιστά τις αφαιρεθέντες εγγραφές. Ανενεργή θέση: περιέχει την defunct εγγραφή.
    - Αναζητούμε την εγγραφή με κλειδί `k`
    - Εάν βρέθει μία τέτοια εγγραφή `(k,v)` την αντικαθιστούμε με την **defunct** εγγραφή και επιστρέφουμε την τιμή `v`
    - Εάν δεν βρεθεί, επιστρέφουμε `null`

Παράδειγμα

- $h(x) = x \bmod 13$



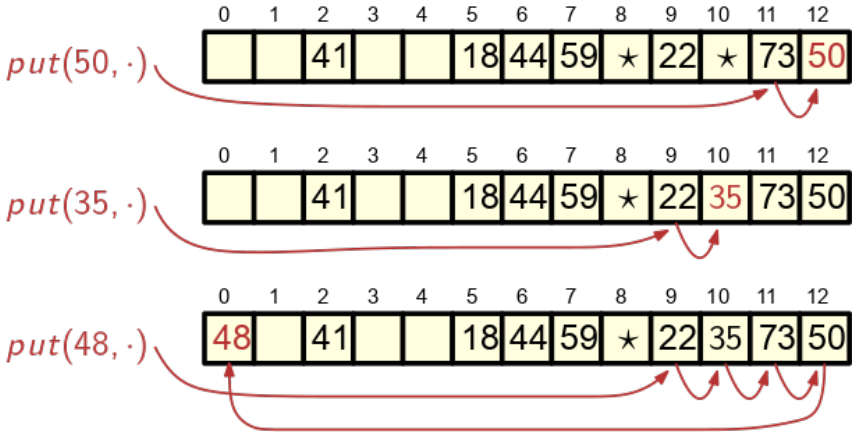
- Επιστρέφει την τιμή της εγγραφής στη θέση 8 και τοποθετεί εκεί την defunct εγγραφή.
- Επιστρέφει την τιμή της εγγραφής στη θέση 10 και τοποθετεί εκεί την defunct εγγραφή.

Πίνακας Κατακερματισμού: Γραμμική εξέταση

- Προσθήκη στοιχείου
  - Εάν ο πίνακας κατακερματισμού είναι γεμάτος, εγείρουμε μία εξαίρεση (exception)
  - Ξεκινάμε την αναζήτηση άδειας θέσης από το κελί $h(k)$
  - Εξετάζουμε διαδοχικά τα κελιά μέχρι να βρεθεί ένα άδειο κελί που να είναι είτε άδειο είτε περιέχει την **defunct** εγγραφή και σε αυτό αποθηκεύουμε την εγγραφή `(k,v)`

Παράδειγμα

- $h(x) = x \bmod 13$



Πίνακας Κατακερματισμού: Γραμμική εξέταση

Ανάλυση

- Θεωρούμε ότι ισχύει η «ομοιόμορφη υπόθεση»

Θεώρημα: Μία επιτυχημένη αναζήτηση σε ένα πίνακα κατακερματισμού με γραμμική διερεύνηση και παράγοντα φόρτου α χρειάζεται, κατά μέσο όρο,

$$\frac{1}{2}\left(1+\frac{1}{1-a}\right)$$

συγκρίσεις, ενώ μία αποτυχημένη

$$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$$

Πίνακας Κατακερματισμού: Τετραγωνική εξέταση

Τετραγωνική εξέταση [quadratic probing]

- Ακολουθία θέσεων που εξετάζονται: $h^k = \{ h(k)$
  $h(k) + 1 \mod N,$
  $h(k) + 2^2 \mod N,$
  $h(k) + 3^2 \mod N,$
  $\vdots$
  $h(k) + j^2 \mod N,$
  $\vdots$
  $\}$
- $h^k(j) = h^k(j-1) + 2j - 1 \mod N$

- Πρέπει να γίνει σωστή επιλογή του μεγέθους του πίνακα N
  - Μπορεί να μην εξετάζονται διαθέσιμες κενες θέσεις

Θεώρημα: Όταν το μέγεθος του πίνακα κατακερματισμού N είναι πρώτος αριθμός και το πλήθος των άδειων θέσεων υπερβαίνει το ήμισυ της χωρητικότητας, τότε είναι πάντοτε δυνατή η εύρεση άδειας θέσης.

Πίνακας Κατακερματισμού: Διπλός κατακερματισμός

Διπλός κατακερματισμός [double hashing]

- Ο διπλός κατακερματισμός χρησιμοποιεί μία δευτερεύουσα συνάρτηση κατακερματισμού $d(k)$ και διαχειρίζεται τις συγκρούσεις τοποθετώντας μία εγγραφή στην πρώτο διαθέσιμη θέση από την ακολουθία:

$$\big(h(k) + j \cdot d(k)\big) mod\ N\ , \qquad j = 0,1, \dots, N-1$$

- Η συνάρτηση $d(k)$ δεν μπορεί να έχει μηδενικές τιμές
- Το μέγεθος N του πίνακα κατακερματισμού πρέπει να είναι πρώτος αριθμός για να είναι δυνατή η διερεύνηση [probing] όλων των θέσεων
- Συνήθης επιλογή για την δευτερεύουσα συνάρτησης κατανομής είναι:

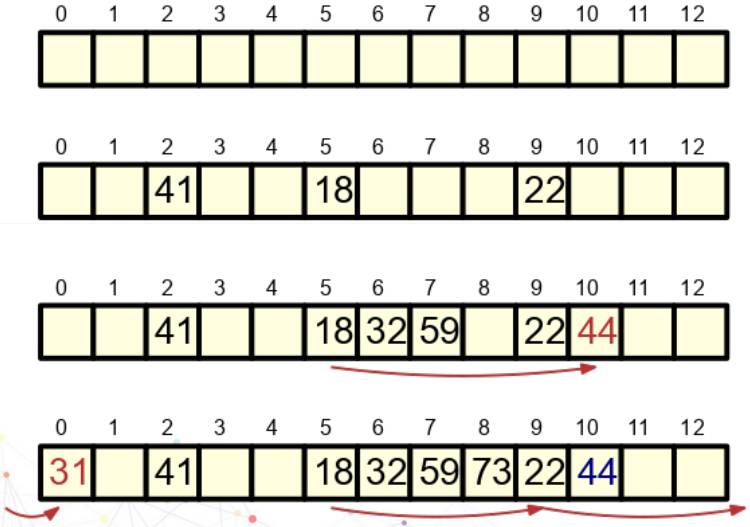
$$d(k) = q - k \bmod q\ , \qquad q < N \quad \text{όπου } q \text{ πρώτος αριθμός}$$

  - οι πιθανές τιμές της $d(k)$ είναι: $1,2, \dots, q$
  - Συνήθης επιλογή: $q = 97$

Πίνακας Κατακερματισμού: Διπλός κατακερματισμός

Παράδειγμα:

- $N = 13$
- $h(k) = k \bmod 13$
- $d(k) = 7 - k \bmod 7$
- Εισαγωγή: 18, 41, 22, 44, 59, 32, 31, 73



k	$h(k)$	$d(k)$	$(h(k) + j \cdot d(k)) \bmod N$
18	5	3	5
41	2	1	2
22	9	6	9
44	5	5	5 → 10
59	7	4	7
32	6	3	6
31	5	4	5 → 9 → 0
73	8	4	8

Πίνακας Κατακερματισμού: Τετραγωνική εξέταση

Ανάλυση

- Θεωρούμε ότι ισχύει η «ομοιόμορφη υπόθεση»

Θεώρημα: Μία επιτυχημένη αναζήτηση σε ένα πίνακα κατακερματισμού με τετραγωνική εξέταση και παράγοντα φόρτου α χρειάζεται, κατά μέσο όρο,

$$\frac{1}{a}\ln\frac{1}{1-a}$$

συγκρίσεις, ενώ μία αποτυχημένη

$$\frac{1}{1-a}$$

Πίνακας Κατακερματισμού: Ανακατακερματισμός

Ανακατακερματισμός [Rehashing]

- Όταν ο παράγοντας φόρτου , λόγω των εισαγωγών, γινει μεγαλύτερος του $\frac{1}{2}$:
  → δέσμευση νέου πίνακα «διπλάσιου» μεγέθους
  → ανακατανομή στο νέο πίνακα

- Όταν ο παράγοντας φόρτου, λόγω των διαγραφών, γίνει μικρότερος του $1/8$:
  → δέσμευση νέου πίνακα «μισού» μεγέθους
  → ανακατανομή στο νέο πίνακα

Θεώρημα: Μία ακολουθία από n ενθέσεις, διαγραφές, και αναζητήσεις σε ένα πίνακα κατακερματισμού απαιτεί χρόνο $O(n)$ και γραμμικό, στο πλήθος των αποθηκευμένων εγγραφών, χώρο.

Πίνακας Κατακερματισμού: Απόδοση

Πολυπλοκότητα:

	Υλοποίηση	
ΑΤΔ Map	Hashing-Chaining	Hashing-Linear Probing
`put(k,v)`	O(1) [AC]	O(1) [AC]
`get(k)`	O(1) [AC]	O(1) [AC]
`remove(k)`	O(1) [AC]	O(1) [AC]
`size()`	O(1)	O(1)
`isEmpty()`	O(1)	O(1)

`entrySet()`
`keySet()`
`values()`
	O(s) [AC]
	s το πλήθος των εγγραφών
		O(s) [AC]
		s το πλήθος των εγγραφών

Πίνακας Κατακερματισμού: Εφαρμογές

Εφαρμογές

- Μικρές βάσεις δεδομένων
- Μεταγγλωτιστές [compilers] (πίνακας συμβόλων)
- Προσωρινή αποθήκευση [cache] φυλλομετρητή [browser]

Διατεταγμένος Χάρτης
[Sorted Map]

Διατεταγμένος Χάρτης [Sorted Map]

- Ο ΑΤΔ Διατεταγμένος Χάρτης (επεκτείνει το Χάρτη)
  - `firstEntry()`: Επιστρέφει την εγγραφή με τη μικρότερη τιμή κλειδιού (ή `null` αν ο χάρτης είναι άδειος)
  - `lastEntry()`: Επιστρέφει την εγγραφή με τη μεγαλύτερη τιμή κλειδιού (ή `null` αν ο χάρτης είναι άδειος)
  - `ceilingEntry(k)`: Επιστρέφει την εγγραφή με το μικρότερο κλειδί από όσα κλειδιά είναι μεγαλύτερα ή ίσα του `k` (ή `null` αν δεν υπάρχει τέτοιο κλειδί)
  - `floorEntry(k)`: Επιστρέφει την εγγραφή με το μεγαλύτερο κλειδί από όσα κλειδιά είναι μικρότερα ή ίσα του `k` (ή `null` αν δεν υπάρχει τέτοιο κλειδί)
  - `lowerEntry(k)`: Επιστρέφει την εγγραφή με το μεγαλύτερο κλειδί από όσα κλειδιά είναι αυστηρά μικρότερα του `k` (ή `null` αν δεν υπάρχει τέτοιο κλειδί)
  - `higherEntry(k)`: Επιστρέφει την εγγραφή με το μικρότερο κλειδί από όσα κλειδιά είναι αυστηρά μεγαλύτερα του `k` (ή `null` αν δεν υπάρχει τέτοιο κλειδί)
  - `subMap(`k_1`,`k_2`)`: Επιστρέφει έναν `iterator` όλων των καταχωρήσεων με κλειδί μεγαλύτερο ή ίσο του k_1 και αυστηρά μικρότερο από το k_2

Διατεταγμένος Χάρτης [Sorted Map]

- Η διαπροσωπία `SortedMap`

```
1  public interface SortedMap<K,V> extends Map<K,V>{
2
3    Entry<K,V> firstEntry();
4    Entry<K,V> lastEntry();
5
6    Entry<K,V> ceilingEntry(K key) throws IllegalArgumentException;
7    Entry<K,V> floorEntry(K key) throws IllegalArgumentException;
8
9    Entry<K,V> lowerEntry(K key) throws IllegalArgumentException;
10   Entry<K,V> higherEntry(K key) throws IllegalArgumentException;
11
12   Iterable<Entry<K,V>> subMap(K fromKey, K toKey) throws IllegalArgumentException;
13 }
```

Διατεταγμένος Χάρτης [Sorted Map]

- Υλοποίηση του διατεταγμένου χάρτη με Sorted ArrayList

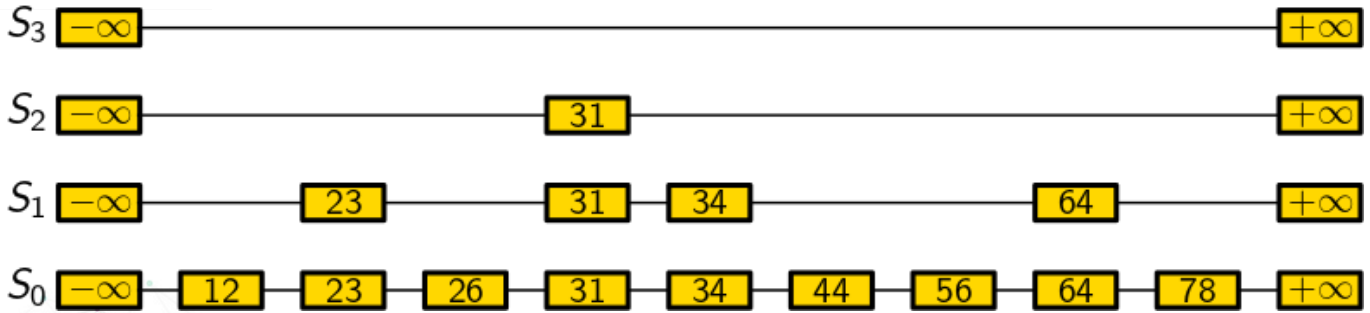
	Υλοποίηση
Μέθοδος	**ArrayList**
`put(k,v)`	`O(n) [O(log n) εάν υπάρχει ήδη το κλειδί]`
`get(k)`	`O(log n)`
`remove(k)`	`O(n)`
`size()`	`O(1)`
`isEmpty()`	`O(1)`
`firstEntry(),lastEntry()`	`O(1)`
`ceilingEntry(k), floorEntry(k), lowerEntry(k), higherEntry(k)`	`O(log n)`
`subMap(k1,k2)`	`O(s + log n), s το πλήθος των στοιχείων μεταξύ k1 και k2`
`entrySet(), keySet(), values()`	`O(n)`

Λίστα Παράλειψης
[Skip List]

Λίστα Παράλειψης [Skip List]

Λίστα παράλειψης [skip list]

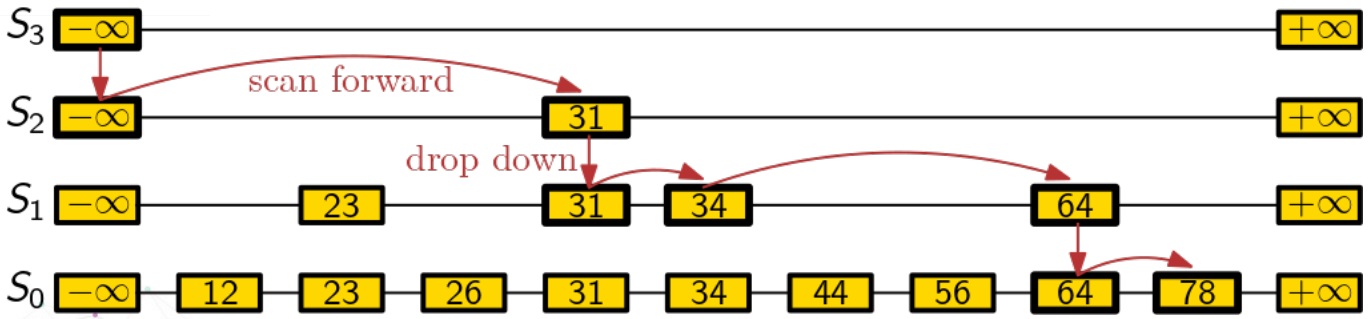
- Μια λίστα παράλειψης για ένα σύνολο S από διακριτές εγγραφές τύπου $(key, element)$ [(κλειδί, στοιχείο)] είναι μια ακολουθία από λίστες $S_0, S_1, \ldots, S_h$ τέτοιες ώστε:
  - Κάθε λίστα S_i περιέχει τα ειδικά κλειδιά $+\infty$ και $-\infty$
  - Η λίστα S_0 περιέχει τα κλειδιά του S σε αύξουσα σειρά
  - Κάθε λίστα είναι μία υποακολουθία της προηγούμενης, δηλαδή: $S_0 \supseteq S_1 \supseteq \cdots \supseteq S_h$
  - Η λίστα S_h περιέχει μόνο τα δύο ειδικά κλειδιά



Λίστα Παράλειψης [Skip List]

- Αναζήτηση στοιχείου
  - Αναζητούμε ένα κλειδί `x` σε μία λίστα παράλειψης, ως ακολούθως:
    - Ξεκινάμε από την πρώτη θέση στην τελευταία λίστα S_h
    - Στην τρέχουσα θέση `p`, συγκρίνουμε το `x` με το `y ← key(next(p))`
      - `x=y`: επιστρέφουμε το `element(next(p))`
      - `x>y`: συνεχίζουμε την έρευνα *μπροστά* [scan forward] (ίδια λίστα)
      - `x<y`: συνεχίζουμε την έρευνα *κάτω* [drop down] (στην προηγούμενη λίστα)
    - Αν προσπαθήσουμε να συνεχίσουμε την έρευνα προς τα *κάτω,* αλλά βρισκόμαστε στη λίστα S_0, επιστρέφουμε `null`

Παράδειγμα
Αναζήτηση του κλειδιού 78



Λίστα Παράλειψης [Skip List]

- Τυχαιοποιημένοι αλγόριθμοι [randomized algorithms]
  - Ένας τυχαιοποιημένος αλγόριθμος εκτελεί ρίψεις νομίσματος για να ελέγξει την εκτέλεση του
  - Περιλαμβάνει εντολές του τύπου:

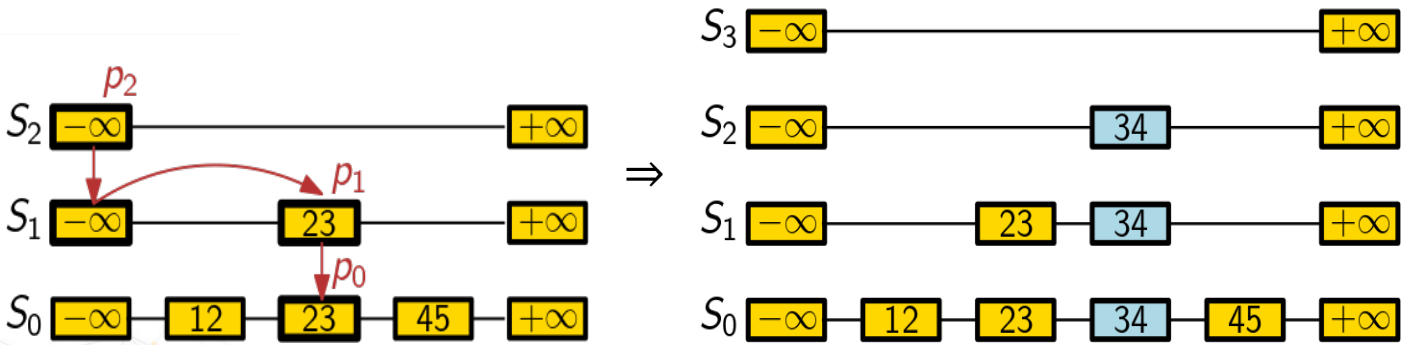
```
b = random()
if b == 0
   do A
else          //b == 1
   do B
```

  - Ο χρόνος εκτέλεσης εξαρτάται από το αποτέλεσμα της ρίψης του νομίσματος
  - Αναλύουμε το προσδοκώμενο χρόνο εκτέλεσης ενός τυχαιοποιημένου αλγόριθμου κάτω από τις ακόλουθες υποθέσεις:
    - Το κέρμα είναι αμερόληπτο
    - Οι ρίψεις του νομίσματος είναι ανεξάρτητες
  - Ο χρόνος εκτέλεσης της χειρότερης περίπτωσης ενός τυχαιοποιημένου αλγόριθμου είναι συχνά μεγάλος αλλά έχει πολύ μικρή πιθανότητα (πχ συμβαίνει όταν όλες οι ρίψεις του νομίσματος δώσουν "κορώνα")

Λίστα Παράλειψης [Skip List]

- Εισαγωγή στοιχείου
  - Για να εισάγουμε μια εγγραφή (x, v) σε μία λίστα παράλειψης, χρησιμοποιούμε ένα τυχαιοποιημένος αλγόριθμο:
    - Ρίχνουμε επαναλαμβανόμενα ένα νόμισμα μέχρι να φέρουμε "γράμματα" και αποθηκεύουμε στη μεταβλητή i πόσες φορές φέραμε "κορώνα"
    - Εάν $i \geq h$, προσθέτουμε στη λίστα παράλειψης τις καινούριες λίστες $S_{h+1}, \ldots, S_{i+1}$, κάθε μία από τις οποίες περιέχει μόνο τα δύο ειδικά κλειδιά
    - Κάνουμε αναζήτηση για το x στη λίστα παράλειψης και βρίσκουμε τις θέσεις $p_0, p_1, \ldots, p_i$ των στοιχείων που έχουν το μεγαλύτερο κλειδί από το σύνολο των κλειδιών που είναι μικρότερα του x σε κάθε λίστα $S_0, S_1, \ldots, S_i$ αντίστοιχα
    - Για $j \leftarrow 0,1, \ldots, i$ εισάγουμε τη εγγραφή (x, v) στη λίστα S_j μετά τη θέση p_j

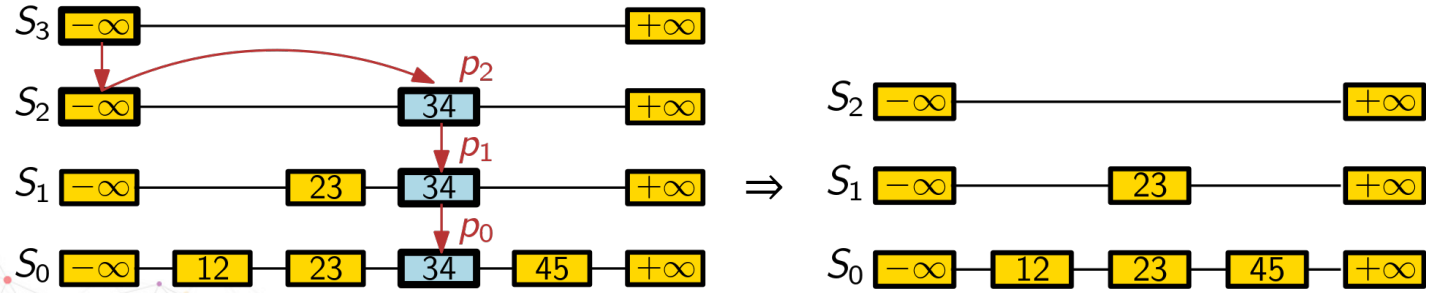
Παράδειγμα:
Εισαγωγή του κλειδιού 34 με $i = 2$



Λίστα Παράλειψης [Skip List]

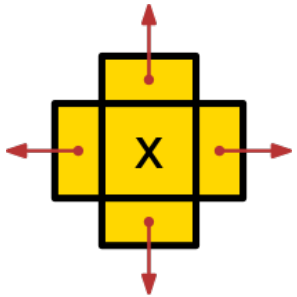
- Διαγραφή στοιχείου
  - Για να διαγράψουμε μια εγγραφή με κλειδί x από μία λίστα παράλειψης, ακολουθούμε τα παρακάτω βήματα:
    - Αναζητούμε το στοιχείο x στη λίστα παράλειψης και βρίσκουμε τις θέσεις $p_0, p_1, \ldots, p_i$ των στοιχείων με κλειδί x σε κάθε λίστα $S_0, S_1, \ldots, S_i$ αντίστοιχα
    - Διαγράφουμε τα στοιχεία των θέσεων $p_0, p_1, \ldots, p_i$ από τις λίστες $S_0, S_1, \ldots, S_i$, αντίστοιχα
    - Διαγράφουμε όλες εκτός από μια από τις λίστες που έχουν μόνο τα δύο ειδικά κλειδιά

Παράδειγμα:
Αφαίρεση του κλειδιού 34



Λίστα Παράλειψης [Skip List]

- Υλοποίηση
  - Μπορούμε να υλοποιήσουμε μια λίστα παράλειψης με κόμβους με 4 συνδέσεις [quad-nodes]
  - Ένας quad-node αποθηκεύει:
    - την εγγραφή (φαίνεται μόνο το κλειδί)
    - μια σύνδεση στον προηγούμενο κόμβο
    - μια σύνδεση στον επόμενο κόμβο
    - μια σύνδεση στον ανώτερο κόμβο
    - μια σύνδεση στον κατώτερο κόμβο
  - Επίσης, ορίζουμε τα ειδικά κλειδιά PLUS_INF και MINUS_INF τροποποιώντας τον comparator των κλειδιών ώστε να μπορεί να τα διαχειριστεί



Λίστα Παράλειψης [Skip List]

- Χρήση μνήμης
  - Ο χώρος που χρησιμοποιείται από μία λίστα παράλειψης εξαρτάται από τις τυχαίες ρίψεις νομίσματος που γίνονται σε κάθε κλήση του αλγορίθμου εισαγωγής
  - Χρησιμοποιούμε δύο βασικά πιθανολογικά γεγονότα:
    - Γεγονός 1: Η πιθανότητα να πάρουμε $\boldsymbol{i}$ συνεχόμενες "κορώνες" όταν ρίχνουμε ένα νόμισμα είναι $\boldsymbol{1/2^i}$
    - Γεγονός 2: Εάν κάθε μία από τις $\boldsymbol{n}$ εγγραφές συμπεριλαμβάνεται σε ένα σύνολο με πιθανότητα $\boldsymbol{p}$, το αναμενόμενο μέγεθος του συνόλου είναι $\boldsymbol{np}$
  - Υποθέτουμε μια λίστα παράλειψης με $\boldsymbol{n}$ εγγραφές
    - Από το γεγονός 1, μία εγγραφή εισάγεται στη λίστα $\boldsymbol{S_i}$ με πιθανότητα $\boldsymbol{1/2^i}$
    - Από το γεγονός 2, το αναμενόμενο μέγεθος της λίστας $\boldsymbol{S_i}$ είναι $\boldsymbol{n/2^i}$
  - Το αναμενόμενο πλήθος των κόμβων στη λίστα παράλειψης είναι:

$$\sum_{i=0}^{h} \frac{n}{2^i} = n \sum_{i=0}^{h} \frac{1}{2^i} < 2n$$

  - Συνεπώς, η αναμενόμενη χρήση μνήμης από μία λίστα παράλειψης με $\boldsymbol{n}$ στοιχεία είναι $\boldsymbol{O(n)}$

Λίστα Παράλειψης [Skip List]

- Ύψος
  - Ο χρόνος εκτέλεσης της αναζήτησης στον αλγόριθμο εισαγωγής επηρεάζεται από το ύψος h της λίστας παράλειψης
  - Δείχνουμε ότι με υψηλή πιθανότητα, μια λίστα παράλειψης με n στοιχεία έχεις ύψος $O(\log n)$
  - Χρησιμοποιούμε το ακόλουθο επιπλέον πιθανολογικό γεγονός:
    - Γεγονός 3: Εάν κάθε ένα από τα n γεγονότα έχουν πιθανότητα p, η πιθανότητα τουλάχιστον ένα γεγονός να συμβεί είναι το πολύ np
  - Υποθέτουμε μια λίστα παράλειψης από n εγγραφές
    - Από το γεγονός 1, εισάγουμε μια εγγραφή στη λίστα S_i με πιθανότητα $1/2^i$
    - Από το γεγονός 3, η πιθανότητα η λίστα S_i να έχει τουλάχιστον ένα στοιχείο είναι το πολύ $n/2^i$
  - Επιλέγοντας το $i = 3\log n$, έχουμε ότι η πιθανότητα η λίστα $S_{3\log n}$ να έχει τουλάχιστον μια εγγραφή είναι το πολύ $n/2^{3\log n} = n/n^3 = 1/n^2$
  - Συνεπώς μια λίστα παράλειψης με n εγγραφές έχει ύψος το πολύ $3\log n$ με πιθανότητα τουλάχιστον $1 - 1/n^2$

Λίστα Παράλειψης [Skip List]

- Χρόνοι εκτέλεσης αναζήτησης και ενημέρωσης
  - Ο χρόνος αναζήτησης σε μια λίστα παράλειψης είναι ανάλογος:
    - του αριθμού των προς τα κάτω [drop down] βημάτων, και
    - του αριθμού των προς τα εμπρός [scan forward] βημάτων
  - Τα προς τα κάτω βήματα [drop down] βήματα είναι φραγμένα από το ύψος της λίστας παράλειψης και συνεπώς είναι $\boldsymbol{O}(\log \boldsymbol{n})$ με υψηλή πιθανότητα
  - Για να αναλύσουμε τα προς τα εμπρός [scan forward] βήματα, χρησιμοποιούμε ένα ακόμη πιθανολογικό γεγονός:
    - Γεγονός 4: Ο αναμενόμενος αριθμός από ρίψεις νομισμάτων που απαιτείται για να προκύψουν "γράμματα" είναι 2

Λίστα Παράλειψης [Skip List]

- Χρόνοι εκτέλεσης αναζήτησης και ενημέρωσης
  - Όταν κάνουμε ένα βήμα προς τα εμπρός [scan forward] σε μια λίστα, τα κλειδιά στα οποία καταλήγουμε δεν ανήκουν σε ανώτερη λίστα
    - Ένα βήμα προς τα εμπρός [scan forward] σχετίζεται με μια προηγηθείσα ρίψη νομίσματος η οποία έδωσε "γράμματα"
  - Από το γεγονός 4, σε κάθε λίστα ο αναμενόμενος αριθμός από βήματα προς τα εμπρός [scan forward] είναι 2
  - Συνεπώς, ο αναμενόμενος αριθμός από βήματα προς τα εμπρός [scan forward] είναι $\boldsymbol{O}(\log \boldsymbol{n})$
  - Καταλήγουμε στο ότι μια αναζήτηση σε μια λίστα παράλειψης χρειάζεται $\boldsymbol{O}(\log \boldsymbol{n})$ αναμενόμενο χρόνο
  - Η ανάλυση της εισαγωγής και της διαγραφής μας δίνει παρόμοια αποτελέσματα

Λίστα Παράλειψης [Skip List]

Πολυπλοκότητα:

	Υλοποίηση
ΑΤΔ SortedMap	Λίστα παράλειψης [skip list]
`put(k,v)`	O(log n) [αναμενόμενο]
`get(k)`	O(log n) [αναμενόμενο]
`remove(k)`	O(log n) [αναμενόμενο]
`size()`	O(1)
`isEmpty()`	O(1)
`firstEntry(),lastEntry()`	O(1)
`ceilingEntry(k), floorEntry(k), lowerEntry(k), higherEntry(k)`	O(log n) [αναμενόμενο]
`subMap(k1,k2)`	O(log n + s) [αναμενόμενο] s το πλήθος των στοιχείων μεταξύ k1 και k2
`entrySet(), keySet(), values()`	O(n)

Λίστα Παράλειψης [Skip List]

- Σύνοψη
  - Μία λίστα παράλειψης είναι μια δομή δεδομένων η οποία χρησιμοποιείται στην υλοποίηση του ΑΤΔ Χάρτης και χρησιμοποιεί έναν τυχαιοποιημένο [randomized] αλγόριθμο εισαγωγής.
  - Η δομή δεδομένων "Λίστα Παράλειψης" παρέχει σχεδόν παρομοίου επιπέδου αναζήτηση σε λίστα με αυτή που παρέχει η δυαδική αναζήτηση [binary search] σε διάνυσμα. Η διαφορά είναι ότι ο απαιτούμενος χρόνος $O(\log n)$ είναι αναμενόμενος με μεγάλη πιθανότητα.
  - Μία λίστα παράλειψης με $\boldsymbol{n}$ καταχωρήσεις αναμένεται να χρησιμοποιήσει $\boldsymbol{O(n)}$ μνήμη
  - Οι λίστες παράλειψης είναι γρήγορες και απλές στην υλοποίηση τους

Σύνολα και Χάρτες με Επαναλήψεις
[Sets and Multimaps]

Σύνολα και Χάρτες με Επαναλήψεις [Sets and Multimaps]

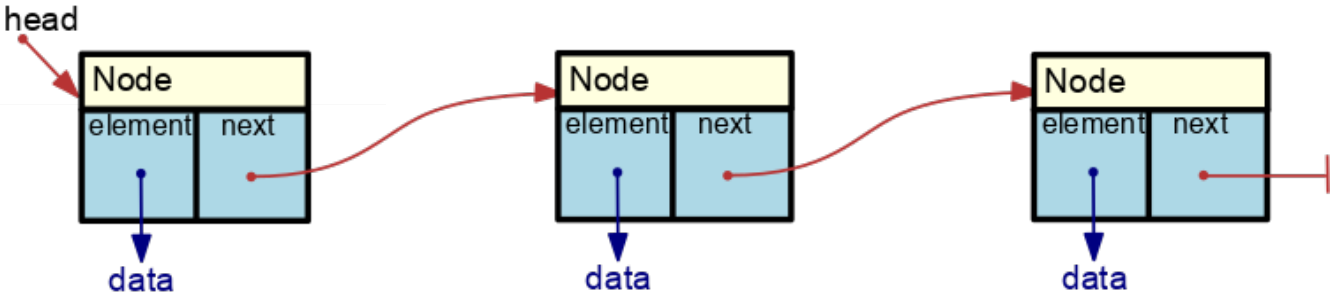
- Ορισμοί
  - Ένα σύνολο [set] είναι μία μη διατεταγμένη συλλογή από στοιχεία, χωρίς διπλότυπα, που τυπικά υποστηρίζει αποδοτικούς ελέγχους για το εάν ένα στοιχείο ανήκει στη συλλογή [membership test]
    - Τα στοιχεία ενός συνόλου μπορεί να θεωρηθούν ως τα κλειδιά ενός χάρτη, αλλά χωρίς τις αντίστοιχες τιμές/δεδομένα που τα ακολουθούν
  - Ένα σύνολο με επαναλήψεις [multiset] (επίσης γνωστό και ως σάκκος [bag]) είναι ένα "σύνολο" το οποίο επιτρέπει διπλότυπα στοιχεία
  - Ένας χάρτης με επαναλήψεις [multimap] (επίσης γνωστό και ως λεξικό [dictionary]) είναι παρόμοιος με ένα κανονικό χάρτη – με την έννοια ότι συσχετίζει τιμές με κλειδιά – όμως διαφέρει στο ότι ένα χάρτης με επαναλήψεις επιτρέπει να συσχετιστεί ένα κλειδί με πολλαπλές τιμές
    - Για παράδειγμα, το ευρετήριο ενός βιβλίου συσχετίζει έναν όρο με μία ή περισσότερες σελίδες στις οποίες αυτός εμφανίζεται

Σύνολα και Χάρτες με Επαναλήψεις [Sets and Multimaps]

- Ο ΑΤΔ Σύνολο
  - `add(e)`: Προσθέτει το στοιχείο `e` στο σύνολο `S` (εάν δεν υπάρχει ήδη)
  - `remove(e)`: Αφαιρεί το στοιχείο `e` από το σύνολο `S` (εάν υπάρχει)
  - `contains(e)`: Ελέγχει εάν το στοιχείο `e` είναι ένα στοιχείο του συνόλου `S`
  - `iterator()`: Επιστρέφει έναν `iterator` των στοιχείων του `S`
  - `addAll(T)`: Ενημερώνει το σύνολο `S` ώστε να περιέχει επίσης όλα τα στοιχεία του συνόλου `T`, αντικαθιστώντας στην πραγματικότητα το σύνολο `S` από το σύνολο `S ∪ T`
  - `retainAll(T)`: Ενημερώνει το σύνολο `S` ώστε να διατηρήσει μόνο τα στοιχεία που ανήκουν και στο σύνολο `T`, αντικαθιστώντας στην πραγματικότητα το σύνολο `S` από το σύνολο `S ∩ T`
  - `removeAll(T)`: Ενημερώνει το σύνολο `S` ώστε να αφαιρέσει όλα τα στοιχεία που ανήκουν στο σύνολο `T`, αντικαθιστώντας στην πραγματικότητα το σύνολο `S` από το σύνολο `S−T`

Σύνολα και Χάρτες με Επαναλήψεις [Sets and Multimaps]

- Αποθήκευση συνόλου σε μία λίστα
  - Μπορούμε να υλοποιοήσουμε ένα σύνολο με μία λίστα
  - Τα στοιχεία αποθηκεύονται ταξινομημένα
  - Χρησιμοποιείται μνήμη μεγέθους $\boldsymbol{O(n)}$



Σύνολα και Χάρτες με Επαναλήψεις [Sets and Multimaps]

Πολυπλοκότητα:

	Υλοποίηση
ΑΤΔ Set	Λίστα [List]
`add(e)`	`O(n)`
`remove(e)`	`O(n)`
`contains(e)`	`O(n)`
`size()`	`O(1)`
`isEmpty()`	`O(1)`
`iterator()`	`O(n)`
`addAll(T)`	`O(n + |T|)`
`retainAll(T)`	`O(n + |T|)`
`removeAll(T)`	`O(n + |T|)`

Σύνολα και Χάρτες με Επαναλήψεις [Sets and Multimaps]

- Χάρτης με επαναλήψεις
  - Ένας χάρτης με επαναλήψεις είναι παρόμοιος με έναν κανονικό χάρτη, μόνο που μπορεί να αποθηκεύσει πολλαπλές εγγραφές με το ίδιο κλειδί (γνωστός και ως λεξικό [dictionary])
  - Μπορούμε να υλοποιήσουμε ένα χάρτη με επαναλήψεις `m` χρησιμοποιώντας ένα κανονικό χάρτη `m'`
    - Για κάθε κλειδί `k` στον χάρτη `m` χρησιμοποιούμε μια λίστα `elements(k)` η οποία περιέχει όλες τις εγγραφές που έχουν ως κλειδί `k`
    - Οι εγγραφές του `m'` είναι ζευγάρια της μορφής `(k, elements(k))`

Σύνολα και Χάρτες με Επαναλήψεις [Sets and Multimaps]

- Ο ΑΤΔ "Χάρτης με Επαναλήψεις" [MultiMap/Dictionary]
  - `get(k)`: Επιστρέφει μια συλλογή από όλες τιμές που αντιστοιχούν στο κλειδί `k` του χάρτη
  - `put(k,v)`: Εισάγει μια νέα εγγραφή `(k,v)` στον χάρτη, χωρίς απώλειες των υπάρχοντων τιμών που αντιστοιχούν στο κλειδί `k`
  - `remove(k,v)`: Αφαιρεί μια εγγραφή που αντιστοιχεί στο κλειδί `k` και έχει την τιμή `v` από το χάρτη (αν υπάρχει κάποια)
  - `removeAll(k)`: Αφαιρεί όλες τις εγγραφές που αντιστοιχούν στο κλειδί `k` του χάρτη
  - `size()`: Το πλήθος των εγγράφων του χάρτη (συμπεριλαμβανομένων και των πολλαπλών τιμών)
  - `entries()`: Επιστρέφει μια συλλογή με όλες τις εγγραφές του χάρτη
  - `keys()`: Επιστρέφει μια συλλογή από τα κλειδιά όλων των εγγραφών του χάρτη (συμπεριλαμβανομένων πολλαπλών εμφανίσεων των κλειδιών για τις αντίστοιχες πολλαπλές τιμές)
  - `keySet()`: Επιστρέφει μια συλλογή από τα μοναδικά κλεδιά όλων των εγγραφών του χάρτη
  - `values()`: Επιστρέφει μια συλλογή από τις τιμές όλων των εγγραφών του χάρτη

Σύνολα και Χάρτες με Επαναλήψεις [Sets and Multimaps]

Πολυπλοκότητα:

	Υλοποίηση
ΑΤΔ MultiMap/Dictionary	Hashing-Linear Probing
put(k,v)	O(1) [αναμενόμενο]
get(k)	O(1) [αναμενόμενο]
remove(k,v)	O(s) [αναμενόμενο] s το πλήθος των εγγραφών με κλειδί κ
size(), isEmpty()	O(1)
removeAll(k)	O(s) [αναμενόμενο]
entries(), keys()	O(n) n το πλήθος των εγγραφών του MultiMap
keySet()	O(m) m το πλήθος των διακριτών κλειδιών του MultiMap
values()	O(d) d το πλήθος των διακριτών τιμών του MultiMap